

EXHIBIT 25

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN JOSE DIVISION

CISCO SYSTEMS, INC.,

Plaintiff,

v.

ARISTA NETWORKS, INC.,

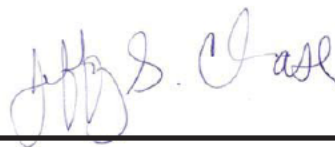
Defendant.

Case No. 5:14-cv-05344-BLF (PSG)

**REBUTTAL EXPERT REPORT OF
JEFFREY S. CHASE, PH.D. REGARDING
NON-INFRINGEMENT OF U.S. PATENT
NO. 7,047,526 AND RELATED MATTERS**

**CONTAINS MATERIAL DESIGNATED HIGHLY CONFIDENTIAL—
ATTORNEYS' EYES ONLY**

Executed on the 17th of June 2016 in Durham, North Carolina.

A handwritten signature in blue ink, appearing to read "Jeffrey S. Chase", is written over a solid black horizontal line.

JEFFREY S. CHASE

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

72. These code segments instantiate the rules of the grammar by creating a Python object to represent each rule. The rule objects are instances of `CliParser.Rule` and its subclasses corresponding to different types of rules.

73. A rule object has methods that allow the parser to test if a given string (a sequence of zero or more tokens) matches the rule. To determine if a rule matches a string, the parser passes tokens from the string one at a time to the Rule object’s `inhale` method, until the rule rejects a token or signals a match by returning a value. If the rule accepts all of the tokens in a string, the parser feeds it a token “None” to prompt it to accept the string as a match or reject the string as incomplete:

```

10 When matching, a rule's 'begin()' function is called. Then, the
11 parser asks the rule to 'inhale()' tokens, one token at a time. The
12 rule can either accept the token (returning any result except
13 the special 'noMatch' value) or refuse
14 the token (returning noMatch). If it refuses the token, it must be
15 prepared to consider other tokens; that is, its state should not
16 change. The rule can also ask subrules to inhale the token. Then,
17 at the end of a phrase, the parser (or parent rule) can ask a
18 subrule to inhale 'None'. This means that we've reached end of file (or
19 at least end of the portion of the file relevant to that rule). The
20 subrule can either accept 'None' (by returning a value) or
21 reject it (by returning noMatch). In either case, the rule will never be
22 asked to inhale anything again until its 'end()' function and then
23 'begin()' function are called.
```

See `CliRule.py` (ARISTA_SRC000074).

74. While a rule is active during parsing (after its `begin` function is called but before its `end` function is called), a context object for the rule exists and is linked to the rule. (*See* `Rule.beginChild` and `Rule.endChild`, ARISTA_SRC000077.) The context object for an active rule contains a context state variable `context.state_`. During parsing, the context state associated

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

with the rule may store information that is relevant to the parser, depending on the rule type, as described below.

C. EOS rule hierarchy

75. Rules are linked together in various hierarchical relationships: a rule may be a subrule of a parent rule. The CLI language is created by combining and composing the rules using subrule relationships in various ways to specify the set of strings to be matched. In effect, each rule (together with its subrules) defines a sublanguage; a string matches a rule if the string is a member of the rule’s sublanguage. The CLI language combines and composes these sublanguages.

76. The rule classes define the nature of each rule’s relationship to its subrules. An OrRule is a rule that matches if any of its subrules matches, and returns the value of the matching subrule on a match. A mode object has an OrRule whose subrules are the registered command rules for the mode. For example, Mode.addCommand installs a new command by invoking Rule.addCommandToRule on the mode’s OrRule. A string of tokens is a valid command for a mode if it matches one of the registered command rules, which are subrules of the mode’s OrRule. Mode.registerShowCommand adds the installed command as a ConcatRule to an OrRule that represents the options to the show command for that mode. The show command is itself a ConcatRule whose sequence is the show command token (e.g., “show”) followed by the OrRule for the options to the show command for that mode. (See ARISTA_SRC000036-37).

77. An OptionalRule matches if one of its subrules matches, or the string is empty. A ConcatRule is a rule that matches if all of its subrules match the tokens of the string in a specific sequence. A TokenRule matches a single token. TokenRule subclasses include KeywordRule, which matches a predefined keyword; PatternRule, which matches any of a set of

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

strings defined by a regular expression; and RangeRule, which matches a number within a specified range. A TokenRule may be created with a DynamicKeywordsMatcher or other kind of dynamic matcher object; such a rule matches against any of multiple tokens present in a dictionary configured for the matcher.

78. At any given time, the context state (`context.state_`) of an active ConcatRule or OrRule includes a list of the rule’s immediate subrules to consider for matches of subsequent tokens. For a ConcatRule, the context state includes an ordered list of references to immediate subrules that must match the remaining tokens in the command string, in order, as a requirement for the ConcatRule to match. For an OrRule, the context state includes a list of references to immediate subrules that have accepted all tokens of the command string consumed so far; for the OrRule to match the command string, at least one of those subrules must match the remaining tokens. I have reviewed Dr. Jeffay’s summary of selected EOS source code pertaining to `context.state_` in paragraphs 101-104 of his report, and I agree with his summary in those paragraphs, except for his statements about how the context for OrRules is created (in 101) and initialized (in 102). However, to the extent that Dr. Jeffay suggests that a context state (or `context.state_`) is a part of any rule, or is contained within or specified or identified by any rule, I disagree. The context state is transient information created and maintained by the parser as it parses each command string with respect to a particular set of stored rules. Although some `context.state_` objects reference associated rule objects, Dr. Jeffay has shown no evidence that any rule object contains, references, specifies, or identifies any associated context state (`context.state_`).

79. Rule objects of these various classes are combined in hierarchies to create more powerful rules and to meet various practical needs for the parser. For example, an OptionalRule

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

is merely an OrRule with an EmptyRule (which matches empty strings) as one of its subrules.

A WrapperRule (such as a HiddenRule) matches if its single subrule matches: the wrapper enables the parser to add some other behavior to the wrapped rule, such as omitting it from help commands (in the case of HiddenRule).

80. Most command rules are ConcatRules. A ConcatRule can match token sequences that constitute complete commands of a given format. Each mode's OrRule also has two KeywordRules registered for the ‘help’ and ‘echo’ commands, which are available in any CLI mode.

81. Typically the code to create and register grammar rules for the EOS CLI uses a syntactic shortcut to specify a ConcatRule as a Python tuple. The rule() method (at ARISTA_SRC000101) converts a tuple expression into a ConcatRule. The elements of the tuple specify an ordered sequence of subrules. If the last element is a callable function, then it is registered as an optional value function for the ConcatRule (see below). This method also translates square-bracketed Python lists into OptionalRules. Tuples and lists may also register a ‘name’ for the specified rule using the ‘>>’ operator.

82. The following code sequence shows examples of rules registered as command rules for a mode:

```

1339 m = EnableMode
1340
1341 tokenEnable = CliParser.KeywordRule( 'enable',
1342                                     helpdesc='Turn on privileged commands' )
1343 tokenPrivLevel = CliParser.RangeRule( 0, 15,
1344                                     helpdesc='Privilege level' )
1345 tokenDisable = CliParser.KeywordRule( 'disable',
1346                                     helpdesc='Turn off privileged commands' )
1347 tokenLogout = CliParser.KeywordRule( 'logout', helpdesc='Exit from EXEC mode' )
1348 tokenQuit = CliParser.KeywordRule( 'quit', helpdesc='Exit from EXEC mode' )
1349 tokenConfigure = CliParser.KeywordRule( 'configure',
1350                                     helpdesc='Enter configuration mode',
1351 guard=CliParser.ssoStandbyGuard )
1352 tokenTerminal = CliParser.KeywordRule( 'terminal',

```

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

```

1353                                     helpdesc='Configure from the terminal' )
1354
1355 m.addCommand( ( tokenConfigure, [ tokenTerminal ] ), m.configure )
1356 m.addCommand( ( tokenEnable, [ '>>privLevel', tokenPrivLevel ], m.enable ) )
1357 m.addCommand( ( tokenDisable, [ '>>privLevel', tokenPrivLevel ], m.disable ) )
1358 m.addCommand( ( tokenExit, ), m.exit, authz=False )
1359 m.addCommand( ( tokenLogout, ), m.exit, authz=False )
1360 m.addCommand( CliParser.HiddenRule( tokenQuit, m.exit ), authz=False )

```

See BasicCli.py (ARISTA_SRC000024). The first five addCommand examples (lines 1355-1359) register newly constructed ConcatRules specified using tuple notation. In 1356-1357, the last element of each ConcatRule tuple is callable, and therefore becomes the ConcatRule’s value function. In 1355 and 1358-1359, no value function is specified, but a function is passed to addCommand, which causes it to be installed as the command rule’s value function. The first element of each ConcatRule in lines 1355-1359 is a TokenRule for a command keyword token (enable, disable, exit, logout). The enable and disable commands include an OptionalRule subrule for an optional privilege level, which is a number between 0 and 15 (see the RangeRule at line 1343). Line 1360 registers a command described by a HiddenRule, which matches a single token ‘quit’, and whose value function invokes the exit function for the mode. All of these rules are examples of conventional grammar rules.

83. This code snippet illustrates that although portions of the rule hierarchy may form a tree, the complete rule hierarchy is not a tree. For example, each of the addCommand lines at 1355-1360 includes a token rule that is a subrule of a ConcatRule (or a HiddenRule), and the addCommand operations register each of the command rules as subrules of the mode’s modeRule OrRule, which is itself a subrule of the mode’s instanceRule OrRule. These rules are organized as a tree. However, in lines 1356-1357, we see that the RangeRule tokenPrivLevel is a subrule of two different OptionalRules: this element has two parents. Thus the rule hierarchy is not a strict hierarchy (*i.e.*, a tree), but rather is an instance of a more general form of data structure called a Directed Acyclic Graph.

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE**D. Rule Values and Value Functions**

84. When a rule matches a sequence of one or more tokens, it returns a value from its `inhale()` method. By default, the value of a rule is a variant of the matching string. For example, the value of a matched `TokenRule` (or a subclass) is the token that the rule matches: it is either the matched token string itself (for a `KeywordRule` or `PatternRule`) or a value that the token represents (e.g., a numeric value for a `RangeRule` that matches a numeric token). The default value of an `OrRule` is the value of its subrule that was matched and selected. The default value of a `ConcatRule` is the value of its last matching subrule.

85. Code that calls a rule object constructor to create a rule object may supply a value function for the rule, as in certain examples listed above. The base `Rule` class provides methods to set and get a rule’s value function, and to invoke the rule’s value function if it is present (`invokeValueFunction`). A rule’s value function, if any, is a callable function referenced by its value function field.

86. If a rule has a value function, then its value is the result of calling its value function when it matches. The `inhale()` method of each rule class calls `Rule.invokeValueFunction()` if the parser calls the rule to inhale a token that completes a match against the rule. If a rule has a value function, then `invokeValueFunction` invokes it and returns its result. If a rule has no value function, then `invokeValueFunction` simply returns a default result passed to `invokeValueFunction` as an argument. (The default result is as discussed above.) In either case, the matched rule’s `inhale` method returns the result of `invokeValueFunction` as the value of the matched rule.

87. Code that registers a rule as a command may supply a value function to `addCommand` or `registerShowCommand`, as shown in certain examples above. In general, a registered command rule has a value function: the rule is either created with a value function, or

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

a value function is specified and added to the rule when it is registered as a command rule, e.g., with addCommand or registerShowCommand.

88. The value function of a command rule executes the command if the command rule matches. In particular, parseAndExecute passes each token of a command string to the inhale method of the current mode's OrRule: if the OrRule returns a match, then this indicates that the command was a valid command. parseAndExecute merely returns the result of a match of a command string with the mode's OrRule: the name 'parseAndExecute' indicates that if a match exists, then the command rule's value function executes the command, and returns the result of executing the command as its value. "An action is the function that is called when a complete CLI command is executed, and carries out the command. An action is really just the value function for the toplevel ConcatRule for the command...." See "Understanding EOS CLI implementation" (<https://eos.arista.com/extending-eos-cli/>).

89. These command actions in the EOS CLI may execute commands in various ways. A few specific examples are presented below. A common pattern for a show command is to read requested information out of Sysdb, which may or may not result in any operation by an agent. Another common pattern is to use the Tac.run primitive to execute a Linux program, which may or may not interact with Sysdb. Dr. Jeffay asserts that "When the CLI executes a value function, the output will be sent to Sysdb. Sysdb then pushes that output to interested agents, which take action in response." Opening Expert Report of Kevin Jeffay, Ph.D. ("Jeffay Report") at ¶¶ 169. This characterization of value functions is inaccurate: many rule value functions act only within the parser, generate no output other than a result that is returned to the parser (from the rule's inhale method), and cause no interaction with Sysdb. Although the value

HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY – SOURCE CODE

functions of command rules may issue requests to Sysdb, many of them do not, and in that case the value functions have no interaction with Sysdb or with agents.

90. I have read Dr. Jeffay’s summary of rule value functions and invokeValueFunction in paragraphs 112-116 of his report, and I agree with the substance of that summary except as noted below. To the extent that Dr. Jeffay characterizes a rule whose value function field is set to ‘None’ as having a value function, I disagree. A rule object whose value function field is set to ‘None’ does not have a value function: ‘None’ is not callable and is not a value function. The value function field is not itself a value function. To the extent that Dr. Jeffay characterizes a call to Rule.invokeValueFunction as an invocation of a value function even for a rule whose value function field is set to ‘None,’ I disagree. A value function value of ‘None’ is not a callable value function and is not invoked by Rule.invokeValueFunction. An invocation of Rule.invokeValueFunction is not itself an invocation of a value function. I note that Dr. Jeffay has provided evidence only that command rules have value functions (in paragraph 114), and not that any other kind of rule has a value function. Dr. Jeffay acknowledges in paragraph 116 that at least some other rules have value function fields set to ‘None.’ To the extent that Dr. Jeffay suggests that a rule value function issues a command to an agent or that the output of a rule value function is written to Sysdb, I disagree. Dr. Jeffay has provided no evidence that any rule value function issues a command to any agent, or that the output of any rule value function is written to Sysdb.

E. Completions

91. The EOS CLI has a capability to identify candidate completions for a partial command string. This capability is not invoked when a user enters a partial command string as a command: if the user does enter an incomplete or ambiguous command string as a command,